Logical agents

Based on slides by S. Russell - http://aima.cs.berkeley.edu/

Thinking and reasoning

♢ Human intelligence:
- processes of **reasoning** that
- operate on internal **knowledge representation**

♢ By analogy, a "logical agent" can:
- use a logical reasoner, which
- infers decisions based on known **facts** and **rules**

The knowledge (facts and rules) are decoupled from the implementation (reasoner).

Example of reasoning

Knowledge:

Rule: $Cloudy \Rightarrow ShouldBringUmbrella$

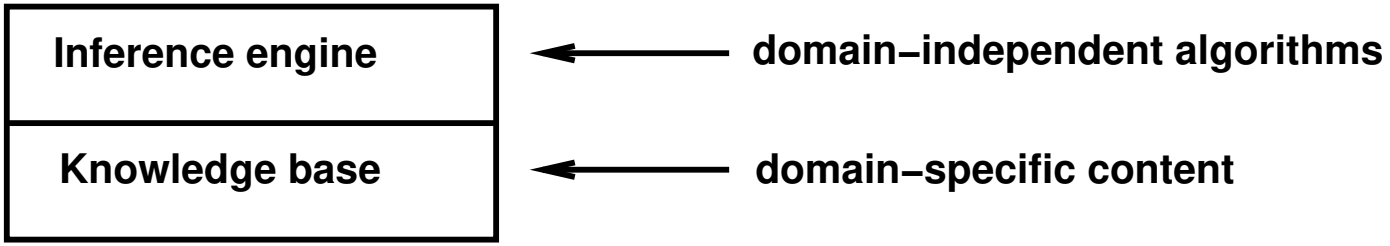
Fact: $Cloudy$

Inference:

We know that it is cloudy, so we can infer that $ShouldBringUmbrella$ is true.

This is new knowledge - it does not exist until we actively infer it from existing knowledge

Knowledge bases



Knowledge base = set of sentences in a **formal** language

Declarative approach to building an agent (or other system):

TELL it what it needs to know

Then it can ASK itself what to do—answers should follow from the KB

Agents can be viewed at the knowledge level

i.e., **what they know**, regardless of how implemented

Or at the implementation level

i.e., data structures in KB and algorithms that manipulate them

Using knowledge bases

Pros:

♢ may be able to mimic human-like decisions

♢ decoupling knowledge from implementation allows people who are not AI experts to develop the knowledge

♢ like databases, can transfer knowledge between systems or upgrade the reasoner to a faster one

Cons:

♢ it is difficult to obtain and maintain knowledge for complex systems

A simple knowledge-based agent

```
function KB-Agent( percept) returns an action
  static: KB, a knowledge base
          t, a counter, initially 0, indicating time

  Tell(KB, Make-Percept-Sentence( percept, t))
  action ← Ask(KB, Make-Action-Query(t))
  Tell(KB, Make-Action-Sentence(action, t))
  t ← t + 1
  return action
```

The agent must be able to:

- Represent states, actions, etc.
- Incorporate new percepts
- Update internal representations of the world
- Deduce hidden properties of the world
- Deduce appropriate actions

Representing knowledge

Knowledge should be:

- easily manageable
- easy to draw new conclusions of

Keep it in chunks called **sentences** of some language:

$x + 2 \geq y$ (arithmetic)

$A \vee \neg A$ (propositional logic)

$\forall\, x, y \;\; Brother(x, y) \;\Rightarrow\; Sibling(x, y)$ (first order logic)

All of the above are sentences that have truth values.

$x + 2 \geq y$ is true if $x = 7,\; y = 1$
$x + 2 \geq y$ is false if $x = 0,\; y = 6$

For now, we will choose **propositional logic** to illustrate basic ideas.

Representing knowledge

Where do the truth values come from?

Take the sentence $A \vee \neg A$.

In all possible worlds (or valuations), A is either true or false.

This sentence happens to be true in all possible worlds.

For $A \vee B$ there are four possible valuations (A and B are either true or false). This sentence is false in some of the possible worlds.

A model is a world with specific truth values assigned to sentences (such as, A is false and B is false).

We say m is a model of a sentence α if α is true in m.

Propositional logic: Syntax

Syntax defines the sentences in the language

The proposition symbols P_1, P_2 etc are sentences

If S is a sentence, $\neg S$ is a sentence (negation)

If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction)

If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction)

If S_1 and S_2 are sentences, $S_1 \Rightarrow S_2$ is a sentence (implication)

If S_1 and S_2 are sentences, $S_1 \Leftrightarrow S_2$ is a sentence (biconditional)

Propositional logic: Semantics

Semantics define the "meaning" of sentences;
i.e., define truth of a sentence

Each model specifies true/false for each proposition symbol

E.g. $P_{1,2}$ $P_{2,2}$ $P_{3,1}$ (8 possible models with 3 symbols)
$true$ $true$ $false$

Rules for evaluating truth with respect to a model m:

$\neg S$	is true iff	S	is false				
$S_1 \wedge S_2$	is true iff	S_1	is true **and**	S_2	is true		
$S_1 \vee S_2$	is true iff	S_1	is true **or**	S_2	is true		
$S_1 \Rightarrow S_2$	is true iff	S_1	is false **or**	S_2	is true		
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$	is true **and**	$S_2 \Rightarrow S_1$	is true		

Simple recursive process evaluates an arbitrary sentence, e.g.,
$\neg P_{1,2} \wedge (P_{2,2} \vee P_{3,1}) = true \wedge (false \vee true) = true \wedge true = true$

Propositional logic: A useful property

Sentences can be either true or false, however:

A sentence is valid (also called a tautology) if it is true in **all** models,
e.g., $True$, $A \vee \neg A$, $A \Rightarrow A$, $(A \wedge (A \Rightarrow B)) \Rightarrow B$

A sentence is satisfiable if it is true in **some** model
e.g., $A \vee B$, C

A sentence is unsatisfiable (or a contradiction) if it is true in **no** models
e.g., $A \wedge \neg A$

If a sentence α is valid then $\neg\alpha$ is unsatisfiable and vice versa.

This is because if α is true in all models, then there are no models left where $\neg\alpha$ could be true – they cannot be true simultaneously.

Asking questions

Entailment (symbol $\models$) means that one thing **follows from** another.

Knowledge base KB entails sentence α
if and only if
α is true in all worlds where KB is true

Assume KB (which is some set of sentences) is not contradictory. Then to find out if α is true, we need to check somehow if $KB \models \alpha$.

Note that α could be true in cases KB is not true. We do not care about these, because the knowledge base represents both our best effort of describing "how things work" and "the current state of things".

Asking questions: Propositional logic

Theorem:

$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

Because $(KB \Rightarrow \alpha) \equiv (\neg KB \vee \alpha)$,

$\neg(\neg KB \vee \alpha)$ must be invalid.

Apply DeMorgan's Law:

$\neg(\neg KB \vee \alpha) \equiv (KB \wedge \neg\alpha)$

So, one idea is that we add the negation of our query $\neg\alpha$ to the knowledge base, and show that it is now **contradictory**.

Asking questions: Propositional logic

Showing that there are **no models** where $KB \wedge \neg\alpha$ is true:

- ♢ Build a complete truth table (always $O(2^n)$).
  There should be no rows where the full sentence evaluates to true.
- ♢ Use a SAT solver with a smart algorithm like DPLL
  (basically finds valuations where a propositional sentence is true).
  If it finds no solutions, there are no models.
- ♢ Use a local search (WalkSAT):
  start with some valuation, flip truth values of individual symbols one by one.
  Not guaranteed to find a model anyway, so the result is a **guess**.

Symbolic proof

We can also try to prove that $(KB \Rightarrow \alpha)$ by applying inference rules to our existing knowledge.

- Can be faster than checking satisfiability
- Can automatically add new facts to the knowledge base

E.g. Modus Ponens:

$$\frac{A \Rightarrow B, \quad A}{B}$$

AND introduction:

$$\frac{A, B}{A \wedge B}$$

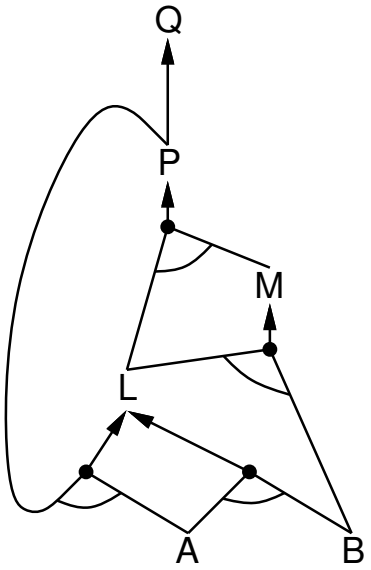
etc.

We can start with what we know, adding new facts one by one. The goal α may eventually be created by some inference rule. This is called forward chaining.

Forward (and backward) chaining

Idea: fire any rule whose premises are satisfied in the KB,
add its conclusion to the KB, until query is found

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



We can also start from the goals (backward chaining) and see if each branch ends with what we already know. Generally, these methods are fast, but require prior conversion of the knowledge base to some normal form.

Expressive power

Propositional logic does not handle generalizations well at all.

Let's try to represent the following knowledge:

Every person is mortal.
Sokrates is a person.
Zeus is a god.

In propositional logic we could write:

$Person \Rightarrow Mortal$
$Sokrates \Rightarrow Person$
$Zeus \Rightarrow God$

We can infer $Mortal$ from this, but what does that mean? Who is mortal and who isn't? In propositional logic, we need explicit symbols like $SokratesIsMortal$.

To describe the knight move rule in chess we would need a set of sentences for each of the 64 squares of the board.

Expressive power

First order logic (FOL) is much more powerful:

$\forall x \;\; Person(x) \Rightarrow Mortal(x)$

$Person(Sokrates)$

$God(Zeus)$

FOL elements:

Constants	$KingJohn,\ 2,\ UCB, \ldots$
Predicates	$Brother,\ >, \ldots$
Functions	$Sqrt,\ LeftLegOf, \ldots$
Variables	$x,\ y,\ a,\ b, \ldots$
Connectives	$\wedge \ \vee \ \neg \ \Rightarrow \ \Leftrightarrow$
Equality	$=$
Quantifiers	$\forall \ \exists$

First order logic: sentences

Brothers are siblings

$\forall\, x, y \;\; Brother(x, y) \;\Rightarrow\; Sibling(x, y)$.

"Sibling" is symmetric

$\forall\, x, y \;\; Sibling(x, y) \;\Leftrightarrow\; Sibling(y, x)$.

One's mother is one's female parent

$\forall\, x, y \;\; Mother(x, y) \;\Leftrightarrow\; (Female(x) \wedge Parent(x, y))$.

First order logic: implementations

Some software for experimenting with first order logic knowledge bases:

♢ PROLOG:
- a programming language, implements FOL partially
- uses: backward chaining
- hundreds of books and websites available

♢ Otter:
- full theorem prover, includes arithmetic etc
- uses: various techniques
- not particularly newbie friendly